Kode Vicious Cycles On

Summer has descended upon us, in all of its hot and sticky glory. And as one of our readers reminds us, too much sun can produce strange effects in koders (and those who write about koding). To escape this fate, many take refuge indoors, within the cool confines of industrial-grade, high-power air conditioning. But it's a false haven. It feels great at first, but prolonged exposure can produce deleterious effects, slowing brain activity as workers slowly go all cryogenic. This is disastrous for koders, who suddenly find themselves stumped by the most basic of tasks. Not to worry, though, Kode Vicious knows this feeling and can administer his unique brand of brain-thawing wisdom to the afflicted. It's sure to knock you right out of your frozen stupor. Contact him at kv@acmqueue.com... and turn that darn AC down!

Kode Vicious received mucho feedback on his "Kode Vicious Reloaded" column, published back in March. In particular, many were concerned with KV's advice to Driven to Abstraction. What follows is a handful of these letters and KV's responses to them.

Dear KV,

I like your column on koding in *ACM Queue*, mostly because at the end it says that you are an "avid bicyclist," just like me. Bicycling in California, where you live, must be so much fun compared to bicycling in Germany, where I live, because of the constantly nice weather you enjoy over there. Too much sun, however, can sometimes make you write columns that seem strange to us Old Europeans. In your "Kode Vicious Reloaded" column (March 2005), your advice to the poor reader, Driven to Abstraction, is "to start with smaller parts of the problem" and then play with them using a scripting language.

Got a question for Kode Vicious? E-mail him at kv@acmqueue.com—if you dare! And if your letter appears in print, he may even send you a Queue coffee mug, if he's in the mood. And oh yeah, we edit letters for content, style, and for your own good!

A koder with attitude, KV ANSWERS

YOUR QUESTIONS.

MISS MANNERS HE AIN'T.

Not only does California give you plenty of sun, it also apparently has employers that give you plenty of time to

play around with the "smaller problems" that you like, in a programming language that's irrelevant to the later implementation. It may be convenient, but it's actually very dangerous to solve your favorite problem first and then add the difficult, not-liked stuff later. Examples: Security—can it be added later? No! Read the pleas of guilty in "Patching the Enterprise" by George Brandman (*ACM Queue*, March 2005: 32-39)! Performance—can it be added later? No! Look into the faces of all the frustrated software reengineers.

Even more frustrating than the lack of sunshine in my country is the absence of employers that give you time to play around with scripting languages when project deadlines impend and budgets are already overspent. We are forced to make good overall designs first and then improve them iteratively. Oh, California—the land of milk and honey and bicycling in the sun!

Koder-User-Rider-Teacher (KURT)

Dear KV,

Regarding your comment to switch from C++ to Python for Driven to Abstraction's "simple" data analysis program, there are two problems with the question/answer scenario you presented:

1. He doesn't understand the problem he is assigned, as he stated confusingly at the end of his letter that "there are just too many things in the system to analyze and each one needs a special case." Thus, it's *not* simple. He has a poor understanding of the problem assigned to him. Statements like that should have come before he wrote any code at all. Had he spent more time mapping out the problem, he could have saved himself some embarrassment.
2. You, however, did an equal disservice by blindly suggesting he switch to another language, without knowing any details of his situation. It sounds like Driven to Distraction was working in the boiler room with

no one else around. The solution, if nothing else, is to teach him how to think and design. I remind you that you are writing for a prestigious international association of professional “thinkers.”

My *first* suggestion is for the developer to do the up-front thinking and design needed so he understands the problem. An interpretive language is no better if he does not understand what the problem is—namely, the characteristics of the data involved and how it needs to be analyzed. Engineers who “bite off more than they can chew” aren’t engineering, they’re *hacking*. You feed the crowd by suggesting nothing more than to hack with a different language. The real problem is not about what language to use but the lack of thought prior to writing code. If my guess is correct, however, teams and managers don’t have time for that. You simply suggested “code as you go” in your answer to Driven to Distraction.

I suggest the IEEE/EAI Standard 12207.0, known as the Development Process.

KH

My Dear Olde Europeans,

I want to thank you both for writing to KV here in the New World where we enjoy plenty of sunshine and benevolent employers. I was actually just enjoying a rubdown from my private masseur here at the office when your letters arrived, but I sent Jacques away so that I could concentrate fully on my answer to you.

What I believe you, unfortunately, missed in my original response was my suggestion that Driven to Abstraction break the problem down into smaller, possibly bite-size chunks. Although it’s nice to think that you could specify all aspects of a program up front, that is the case only if all of the parts of the problem are understood before you start designing. I’m sure that each of you has been confronted with a system that you did not completely understand, and that to be able to get a handle on the issues, you had to work with smaller models and prototypes to get your mind around the problem and finally solve it.

It’s as easy to waste huge amounts of time over-specifying a system as it is to waste time playing with a scripted prototype of a sub-problem, and each is equally dangerous to the success of a project. What is most dangerous, though, is simply staring at the same blank screen, page of your notebook, or whiteboard day after day without making any real progress. Telling your boss, “Well, I spent the last week thinking about the problem,” is not acceptable, even here in the Land of Milk and Honey.

My suggestion was meant to break the mental stalemate that Distracted had gotten into, the Zen equivalent

of a tweak on the nose or a tap with a stick. I figure telling people to break down larger problems into smaller problems is more acceptable in the workplace than wandering around tweaking their noses or hitting them with sticks.

KV

Dear KV,

While I agree it’s not a good idea to bite off more than you can chew, I’m concerned with your response to Driven to Abstraction’s question. My fear is that your faithful readers will think it’s OK *not* to create classes. I’ve seen too many supposedly object-oriented programs composed of a single class. I realize you don’t suggest not adding classes, but you also do not directly address Driven to Abstraction’s fear of classes.

Afraid of Those Afraid of Classes

Dear ATAC,

It’s nice to see another take on the problem with my response to Driven to Abstraction, and with this one I can wholeheartedly agree. Just two days ago I was reviewing some code that was clearly written by someone either afraid of or totally ignorant of classes. Actually, this koder seemed also to be ignorant of the concept of modularity, as everything was in a single, 5,000-line file. Internally, the code was very clever, but in its current state totally un reusable. It now falls to KV to make the kode a bit klassier.

Unfortunately, this is a common problem I suspect we all face. Either because of time pressure or lack of training, koders decide to bite off not only more than they can chew, but also more than anyone else can swallow.

As with many things in the world, there is a spectrum between too big and too small. Too often, people kode only for themselves without realizing that everything they create must be read and debugged by others. If we could only mend our selfish ways, perhaps we could all just get along.

KV

KODE VICIOUS, known to mere mortals as George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor’s degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who has made San Francisco his home since 1990.

© 2005 ACM 1542-7730/05/0700 \$5.00